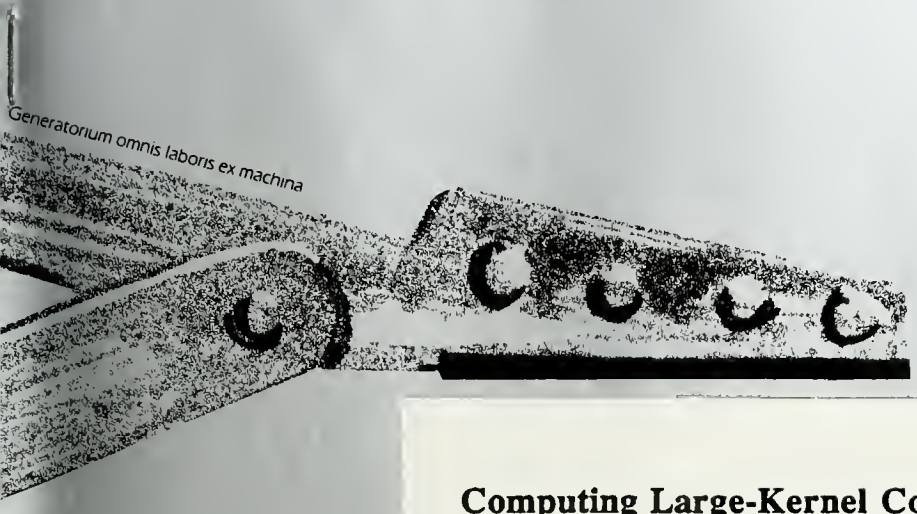
Robotics Research Technical Report



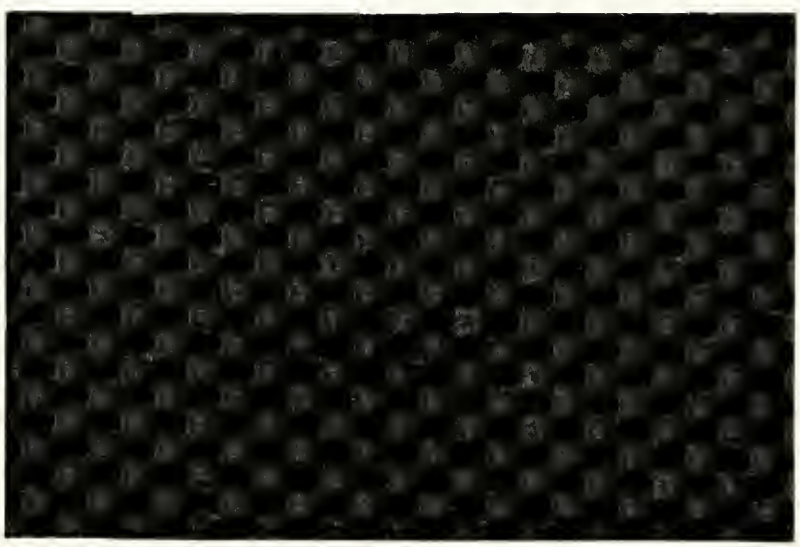
Computing Large-Kernel Convolutions of Images

by

Robert Hummel
David Lowe
Courant Institute of Mathematical Sciences

Technical Report No. 254
Robotics Report No. 84
November, 1986

New York University
Courant Institute of Mathematical Sciences
Computer Science Division
Mercer Street New York, N.Y. 10012



Computing Large-Kernel Convolutions of Images

by

Robert Hummel
David Lowe
Courant Institute of Mathematical Sciences

Technical Report No. 254
Robotics Report No. 84
November, 1986

New York University
251 Mercer Street
New York, NY 10012

This research was supported by Office of Naval Research Grant N00014-85-K-0077, Work Unit NR 4007006, and NSF grants DCR-8403300 and DCR-8502009.

Computing Large-Kernel Convolutions of Images

Robert Hummel and David Lowe

Courant Institute of Mathematical Sciences
New York University
251 Mercer Street, New York, NY 10012 USA

Abstract

Blurring by Gaussian convolution, or by a Laplacian of a Gaussian kernel, is a common image processing technique used for edge detection, multiresolution representation, motion analysis, matching, and other early visual processing tasks. We consider methods for computing Gaussian convolutions on discrete grids, and also consider difference-of-Gaussian and Laplacian-of-Gaussian convolutions. To properly approximate the continuous theory, it is important to evaluate the elements in these large kernels by integration of the continuous kernel in a region about the corresponding location. We discuss complexities of implementations of Gaussian and Laplacian-of-Gaussian kernels on different serial and parallel computer architectures. We further consider zero-crossings of filtered images, and look at methods for selecting significant zero-crossing curves. Finally, we discuss how to handle borders.

1. Continuous Convolution

Many image processing operations rely on convolutions against the image intensity data. While many operators are local, and can be computed using convolutions with three-by-three pixel kernels, there are some operators that require much larger convolution kernels. For example, the Marr-Hildreth edge operator [1] requires convolution kernels that are sometimes as large as fifty by fifty pixels in extent. Most of the large kernel filters are based on Gaussian blurring of the image data either to compute a blurred, smoothed version of the image data, or to compute a difference-of-Gaussian image, or a Laplacian-of-Gaussian image.

We will give a number of computational techniques needed for efficient, accurate digital implementation of these kernels and interpretation of the results. We will also analyze the computational complexity of these implementations, using several models of computation. Specifically, we consider a sequential random access computer, a pipeline parallel machine with a finite but large word size, and a bit-serial mesh-connected parallel computer. Table 1 shows the models of computation, properties, and types of machines that we have in mind. Many of the computational methods discussed here are well-known, and we are mainly concerned with bringing together a compendium of observations in implementing large kernels based on Gaussians, and comparing their complexities on different architectures. A more thorough treatment of large kernel convolutions, together with many further results, can be found in [2]. A recent article by Huertas and Medioni [3] also

Model	Description	Unit-time Operations	Examples
SRAM	Sequential random access machine; uniprocessor	Add,multiply,fetch, store of words (large word size).	VAX, PC
Pipeline	Image processing pipeline machine; raster-scan machine; vector machine.	Table-look-up; add of two images,multiply of image by constant, scroll (sometimes zero-cost).	VICOM DeAnza, CRAY XMP
MCC	Bit-serial mesh-connected computer	Sum or multiply of two 1-bit images,access neighbor bit, 1-bit image shift.	MPP, GAPP

Table 1. Three models of computation based on three different computer architectures.

contains many relevant suggestions for implementation of the types of kernels discussed here.

A Gaussian convolution is defined, in two continuous variables, by

$$h(x,y) = \int_{\mathbb{R}^2} G(x-x',y-y')f(x',y')dx'dy',$$

where

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}.$$

Here $f(x,y)$ is the input image, and $h(x,y)$ is the desired filtered image. In this paper, we address the following questions:

(1) How can the computation be discretized?

(2) How can the discretized equation be computed efficiently?

(3) How should the borders be handled? That is, since $f(x,y)$ is not necessarily defined over all of $\mathbb{R}^2$, the convolution must be converted to a finite domain.

The need for discretization is imposed since the input function is in fact given at a discrete grid of points, say $f(i,j)$, where i and j take on integer values. The temptation is to convert the integral into a sum, replacing the integrand by point evaluations at the grid points, i.e.,

$$h(i,j) \approx \sum_{i'}\sum_{j'} G(i-i',j-j')f(i',j') \quad .$$

This can lead to inaccuracies, however.

Instead, for all convolutions, including Gaussian convolution, the integration should be discretized by modeling the continuous $f(x,y)$ from the given discrete data $f(i,j)$. The easiest method is to assume that $f(x,y)$ is obtained from $f(i,j)$ by nearest-neighbor interpolation. Then the continuous integral can be computed from the discrete sum in the form:

$$h(i,j) = \sum_{i'} \sum_{j'} \bar{G}(i-i', j-j') f(i', j'),$$

where

$$\bar{G}(i,j) = \int_{i-1/2}^{i+1/2} \int_{j-1/2}^{j+1/2} G(x,y)\,dx\,dy.$$

Thus the discrete kernel elements should be obtained from block averages of the continuous kernel, and not by point evaluations.

If bilinear interpolation is used to model $f(x,y)$ instead of nearest neighbor interpolation, the formula becomes:

$$\bar{G}(i,j) = \int_{i-1}^{i+1} \int_{j-1}^{j+1} G(x,y) \cdot (1 - |i-x|) \cdot (1 - |j-y|)\,dx\,dy.$$

Higher order interpolants can be treated similarly. In general, the discrete kernel elements are obtained by integrating the continuous kernel against the basis functions of the interpolation method.

The result of the block averaging procedure is that the kernel values can be changed somewhat. Figure 1 shows a one-dimensional Gaussian curve with dots at the integer points, superimposed on a bar graph representing the block average values. The bar graph gives, in a sense, a more accurate discrete Gaussian convolution. Further, the (infinite) sum of the averaged values $\bar{G}(i,j)$ are guaranteed to sum to one. Interestingly, the infinite sum of the evaluates of the Gaussian $G(i,j)$ also very nearly sum to one, as long as σ is larger than about 0.5 pixels. This occurs because the errors tend to cancel, due to the fact that the Gaussian has a region of negative curvature, and another region with positive curvature. The first row of Table 2 gives the sum of the kernel values for different values of σ for a two-dimensional Gaussian kernel. However, the fact that the point evaluates of the Gaussian very nearly sum to one doesn't justify their use for Gaussian convolution — the averaged values $\bar{G}$ form a better approximation to the continuous theory.

Indeed, assume that the image data is bounded, say $|f(x,y)| \leq M$. We have denoted the convolution using block average kernel values $\bar{G}(i,j)$ by $h(i,j)$. If point evaluates $G(i,j)$ are used instead, let us denote the result by $h_P(i,j)$:

$$h_P(i,j) = \sum_{i'} \sum_{j'} G(i-i', j-j') f(i', j') .$$

We can then bound the difference:

$$|h(i,j) - h_P(i,j)| \leq \sum_{i'} \sum_{j'} |G(i',j') - \bar{G}(i',j')| \cdot M .$$

The bound can be attained, and thus the sum of absolute differences of the kernel elements,

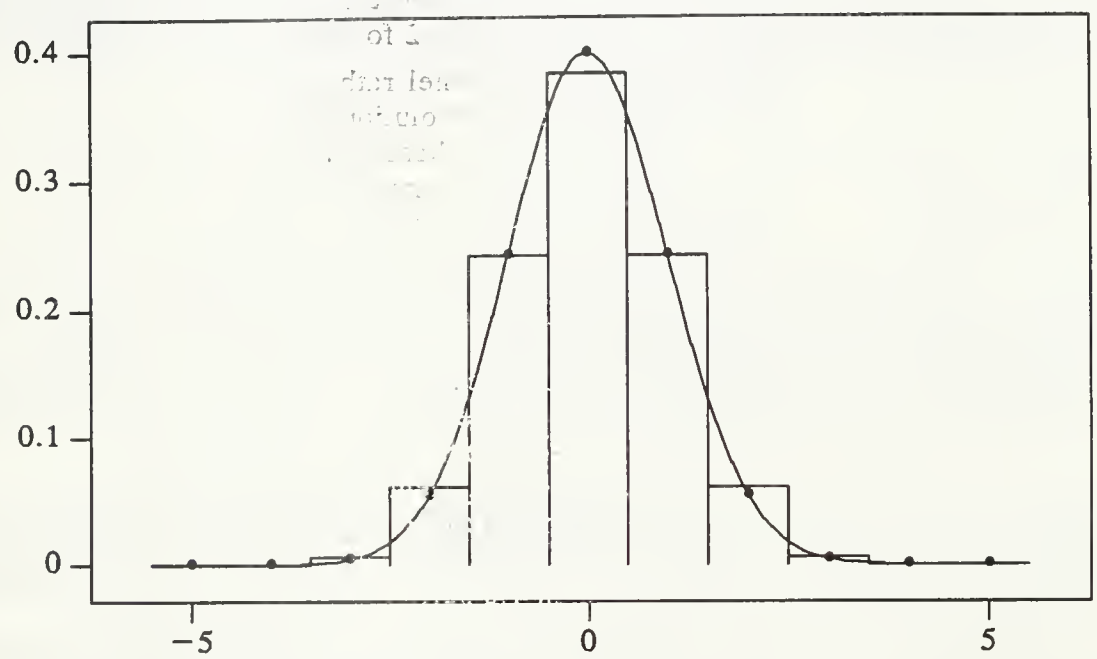


Figure 1. A one-dimensional Gaussian, with $\sigma = 1.0$. Dots mark point evaluates of the Gaussian at each integer. The heights of the bars in the bar graph give the mean values of the Gaussian in a block of width 1 centered at each integer.

Sum	σ								
	.50	.60	.75	1.0	1.5	2.0	3.0	4.0	5.0
$\sum_i \sum_j G(i,j)$	1.02897	1.00328	1.00006	1.0	1.0	1.0	1.0	1.0	1.0
$\sum\sum \lvert G(i,j) - \overline{G}(i,j) \rvert$	0.31213	0.17204	0.09651	0.058	0.026	0.015	0.0068	0.0038	0.0024
$\sum_i \sum_j \Delta G(i,j)$	-1.15203	-0.12971	-0.00238	~ 0.0	~ 0.0	~ 0.0	~ 0.0	~ 0.0	~ 0.0
$\sum\sum \lvert \Delta G(i,j) - \overline{\Delta G}(i,j) \rvert$	3.84728	1.86594	1.17351	0.768	0.414	0.225	0.1039	0.0593	0.0382

Table 2. The sum of point-evaluate kernel elements, where $G(x,y)$ is a Gaussian in two variables with standard deviation σ and total integral 1.

$$\sum_i \sum_j |G(i,j) - \bar{G}(i,j)| ,$$

measures a proportionality factor for the maximum error in using point evaluates. These values are listed in the second row of Table 2 for various values of σ.

The need for averaging the continuous kernel rather than point evaluations is more critical when Laplacian-of-Gaussian convolutions are attempted. We will shortly discuss alternate methods of performing these convolutions, but sometimes it is desired to perform a full image convolution against the analytically calculated kernel. In the case of Laplacian-of-Gaussian convolution, the kernel is given by:[1]

$$\Delta G(x,y) = C \cdot \left(\frac{x^2+y^2}{\sigma^2} - 2 \right) e^{-(x^2+y^2)/2\sigma^2} .$$

Here C is a constant, which if a precise Laplacian-of-Gaussian is desired is equal to $1/(2\pi\sigma^4)$. Once again, point evaluations of the kernel lead to different values than block averages. Figure 2 depicts the one-dimensional situation with the Laplacian of the Gaussian, with $\sigma = 1.0$. This time, kernel elements should sum to zero. Once again, the block-averaged kernel elements, $(\overline{\Delta G})(i,j)$, are guaranteed to sum to zero, while the point evaluates $\Delta G(i,j)$ serendipitously very nearly sum to zero, as long as σ is sufficiently large (greater than 1.0 suffices). The third row of entries in Table 2 gives the values for the infinite sum of kernel elements. The error bound for using point evaluates of the Laplacian-of-Gaussian as opposed to block averages can be measured by the sum of absolute differences:

$$\sum_i \sum_j |\overline{\Delta G}(i,j) - \Delta G(i,j)| .$$

These values are listed in the fourth row of Table 2 for varying values of σ. We see that point evaluates can lead to rather large errors, for example, as high as 10% relative error for $\sigma = 3$ pixels.

Analytic evaluation of the averages of the kernels is possible in terms of the error function *erf*. For example, block averages of the Gaussian are given by:

$$\bar{G}(i,j) = \int_{i-1/2}^{i+1/2} \int_{j-1/2}^{j+1/2} G(x,y)\,dx\,dy =$$

$$\frac{1}{4}\left[\mathrm{erf}\left(\frac{i+1/2}{\sqrt{2}\sigma} \right) - \mathrm{erf}\left(\frac{i-1/2}{\sqrt{2}\sigma} \right) \right] \cdot \left[\mathrm{erf}\left(\frac{j+1/2}{\sqrt{2}\sigma} \right) - \mathrm{erf}\left(\frac{j-1/2}{\sqrt{2}\sigma} \right) \right],$$

where *erf* is the error function associated with the normal distribution, i.e., $\mathrm{erf}(x) = \int_0^x (2/\sqrt{\pi}) e^{-t^2} dt$. (*Erf* is a built-in function on most Fortran compilers). Similarly, block averages of the Laplacian-of-Gaussian kernel are given by

[1]We use the notation " Δ " to denote the Laplacian operator, as is common in mathematical analysis. In physics, the notation " ∇^2 " is often used for the Laplacian operator in $\mathbb{R}^3$ (three-dimensional space), as a shorthand for the divergence of the gradient. The notation is explained by the fact that the gradient of a function f is generally denoted by ∇f, and the divergence of the result is denoted $\nabla \cdot \nabla f$. The notation " ∇^2 " has been borrowed and applied in $\mathbb{R}^n$ for more general n by many disciplines.

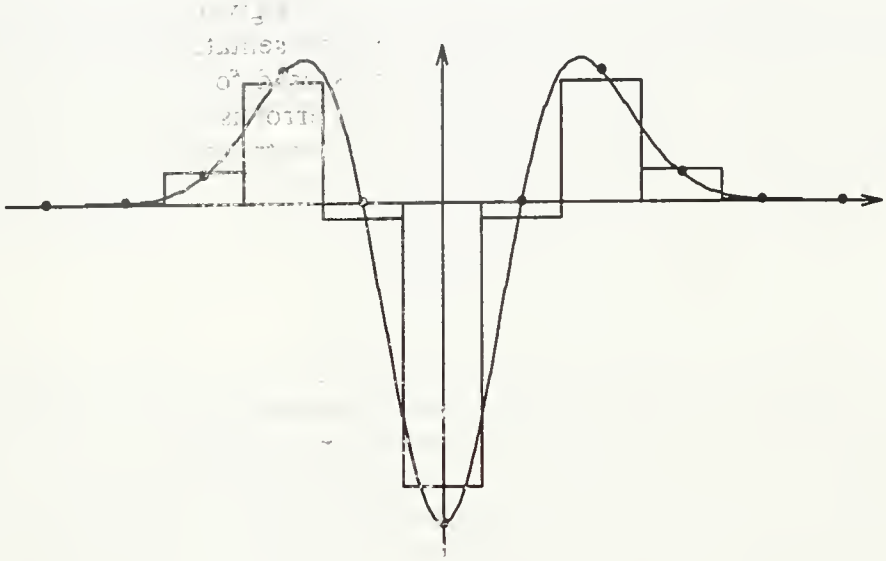


Figure 2. The Laplacian of a Gaussian in one dimension, i.e., the second derivative of a Gaussian, with $\sigma = 1.0$. The point evaluates and mean values at each integer are shown as in Figure 1.

$$\overline{\Delta G}(i,j) = \int_{i-1/2}^{i+1/2}\int_{j-1/2}^{j+1/2} \Delta G(x,y)\,dx\,dy =$$

$$\frac{1}{2}\left[g'(i+1/2) - g'(i-1/2)\right]\left[\operatorname{erf}\left(\frac{j+1/2}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{j-1/2}{\sqrt{2}\sigma}\right)\right]$$

$$+ \frac{1}{2}\left[g'(j+1/2) - g'(j-1/2)\right]\left[\operatorname{erf}\left(\frac{i+1/2}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{i-1/2}{\sqrt{2}\sigma}\right)\right] ,$$

where $g'(x) = (-x/\sqrt{2\pi}\sigma^3)e^{-x^2/2\sigma^2}$.

For more general continuous kernels, analytic evaluation may be impossible or simply too much of a bother. The averaged kernel can then be computed (off-line) by numerical quadrature methods, using standard numerical integration package software. Simpson's method for one-dimensional integration usually suffices for the kernels under consideration. More sophisticated techniques can also be used.

A separate issue when implementing these convolutions concerns quantization and truncation of the kernel. In practice, the kernel elements are typically approximated by rational numbers, so that integer arithmetic may be used. Depending on the number of bits used to represent the kernel elements, the kernel will be effectively truncated to a finite domain, since elements outside of some radius will be

approximated by zero. Further truncation is also possible. When using these kernels, errors come from two separate sources: from the quantization error due to the approximation of the kernel elements by rational values, and the truncation error due to the omission of kernel elements outside of a given radius. The first error is controlled by the number of bits used in the representation of the kernel elements. The truncation error is dependent on the radius used for the implementation of the kernel. For bounded image data, the truncation error is bounded by an amount proportional to the integral of the kernel outside of the truncation radius.

Hildreth notes that the central negative region in $\Delta G(x,y)$ has radius $w=\sqrt{2}\sigma$, and then suggests that the Laplacian-of-Gaussian kernel be implemented with a radius of $4w \approx 5.66\sigma$ [4]. In Table 3, we show the integrals of the continuous kernels $G(x,y)$ and $\Delta G(x,y)$ to various truncation radii $t\cdot\sigma$. (Actually, the table gives the integrals of the kernels in a box of size $2t\sigma$ by $2t\sigma$). We also give the magnitude of the kernel values at the outer radius location. It should be noted that this table is independent of σ. The differences between the integral values over the finite domain and the integrals over the infinite domain give estimates of the truncation error. The quantization error is harder to estimate, but can be made small by using many bits in the representation of the kernel. The magnitude of the kernel at the truncation error gives an indication of the minimum number of bits required. For convolution against a Gaussian, a kernel size out to a radius of 3.5σ yields an accuracy to within .001 times the image data bound, and will require kernel elements with 12 bits. To achieve a .0001 accuracy, a radius of 4.1σ is needed, and kernel elements will require at least 16 bits. For convolution against the Laplacian-of-Gaussian kernel, .001 accuracy is achieved with a 4.2σ truncation radius, requiring 12 bit kernel elements, while .0001 accuracy yields a 4.8σ radius and 16 bits. At the radius 5.66σ, the truncation error for Gaussian convolution is infinitesimal, and the kernel elements at the truncation radius have magnitude roughly 10^{-8}, so that 32 bits for the kernel elements is not unreasonable. For Laplacian-of-Gaussian

Value	t					
	1.	2.	3.	4.	5.	6.
$\int_{-t\sigma}^{t\sigma}\int_{-t\sigma}^{t\sigma} G(x,y)dxdy$	0.466065	0.911070	0.994608	0.999873	0.999999	1.00000
$\int_{-t\sigma}^{t\sigma}\int_{-t\sigma}^{t\sigma} \Delta G(x,y)dxdy$	-0.660764	-0.412275	-5.30386e-02	-2.14115e-03	-2.97344e-05	-1.45821e-07
$\lvert G(t\sigma,0)\rvert$	9.65324e-02	2.15393e-02	1.76805e-03	5.33905e-05	5.93115e-07	2.42393e-09
$\Delta G(t\sigma,0)$	-9.65324e-02	4.30786e-02	1.23764e-02	7.47468e-04	1.36417e-05	8.24135e-08

Table 3. Values for the integrals of a Gaussian and Laplacian-of-Gaussian in a box of increasing size. We also give the value of these functions at the truncation location nearest the origin.

convolution, a radius of 5.66σ will give roughly one part in 10^6 accuracy, and kernel elements with at least 21 bits should be used.

Of course, high accuracy is an issue only if one believes that it is important that the appropriate kernel (Gaussian or Laplacian-of-Gaussian) be implemented accurately. For example, using point evaluates for the kernel elements leads to potentially different results, and thus is not necessarily an accurate implementation. However, it might happen that for the intended application, the differences are inconsequential, and the Gaussian or Laplacian-of-Gaussian can be replaced by more general kernels.

The complexity of pointwise convolutions, without making use of separable kernels or other decomposition methods (discussed below), is a simple matter. On a uniprocessor, the convolution of an N by N image with an m by m kernel will take time $O(m^2N^2)$, or $O(m^2)$ time per pixel. A pipeline machine will frequently have special-purpose hardware for performing, say, p by q convolutions in one pass through the data. In this case, $O(m^2/pq)$ passes will be needed. Finally, on a bit-serial mesh-connected computer, m^2 multiply-accumulates are needed, with m^2 shifts of the image data. Each multiply-accumulate takes a fixed amount of time, depending on the number of bits used to represent the kernel elements and the number of bits used to store the results. Suppose that the accumulates contain b bits. Then the whole process is $O(m^2b)$ complexity.

Rather than converting these asymptotic complexities into typical running time estimates, we defer further calculations until the end of the next section. In that section, we discuss a couple of computational short-cuts that have a profound effect on both the asymptotic complexities and the expected running times.

2. Iterated Discrete Convolution

Two computational tricks permit rapid discrete Gaussian convolution. The first trick, almost always used in practice, takes advantage of the fact that the Gaussian is a separable, symmetric, kernel. This has the consequence that a two-dimensional convolution can be performed by iterating two one-dimensional symmetric convolutions. That is, when the kernel $K(i,j)$ is separable, given say by

$$K(i,j) = k_1(i)\cdot k_2(j),$$

then the convolution can be performed according to the formula

$$\sum_{i'}\sum_{j'} K(i-i',j-j')f(i',j') = \sum_{i'} k_1(i-i')\left[\sum_{j'} k_2(j-j')f(i',j')\right].$$

Thus the horizontal convolution against k_2 is performed first, and the result is convolved against the vertical kernel k_1. If the kernel is m by m, then this observation reduces the complexity from $O(m^2)$ per pixel to $O(m)$. For the Gaussian kernel, both the continuous and discrete versions are separable. Further computational savings can be realized on a sequential machine by making use of the symmetry of the kernels. Specifically, the one-dimension convolutions

$$\sum_{i'} k_1(i-i')f(i')$$

can be implemented as

$$k_1(0)f(i) + \sum_{i' \geq 1} k_1(i')(f(i-i') + f(i+i')) ,$$

since $k_1(-l)=k_1(l)$. This trick saves a few multiplies, but is generally not helpful on a mesh-connected parallel architecture. In Tables 4, 5 and 6 we refer to the use of the symmetry and separability of the kernel as computation Method A.

The second trick applicable to Gaussian convolution makes use of the fact that the binomial coefficients form a rapid approximation to a normal distribution. It is well-known that the binomial coefficients, given by

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

approximate a normal distribution

$$\frac{1}{\sqrt{2\pi}\sigma} e^{x^2/2\sigma^2}$$

where $x = k-n/2$, and $\sigma = (\sqrt{n}/2)$. Binomial coefficients are most easily calculated using Pascal's triangle, and lead to the design of an iterated convolution kernel, formed as follows. The initial data is $f(i)$, $i = \cdots, -1, 0, 1, \cdots$, (in one dimension), which we set equal to $h_0(i)$. We define $h_N(i)$, for $N>0$, by

$$h_N(i) = \frac{h_{N-1}(i-1)+2h_{N-1}(i)+h_{N-1}(i+1)}{4}.$$

It follows that

$$h_N(i) = \sum_{k=-N}^{N} \frac{1}{2^{2N}} \binom{2N}{k+N} f(i+k).$$

This is a close approximation (for large N) of convolution against the Gaussian with $\sigma = \sqrt{N/2}$. This method of Gaussian convolution has the advantage of yielding a

	σ=1			σ=2			σ=4		
	ops	ac	sz	ops	ac	sz	ops	ac	sz
Without Methods A or B, 4σ cutoff	162	81	(9x9)	578	289	(17x17)	2178	1089	(33x33)
Method A, 4σ cutoff	28	20	(9x9)	52	36	(17x17)	100	68	(33x33)
Method A, 6σ cutoff	42	28	(13x13)	76	52	(25x25)	151	104	(51x51)
Method B	12	16	(5x5)	48	64	(17x17)	192	256	(65x65)

Table 4. Serial Random-Access computer model: number of arithmetic operations (ops) and number of memory accesses (ac) per image pixel point for Gaussian convolution using various methods of computation for various sizes of Gaussians. The effective size of the 2-D convolution is given in the size (sz) column. Generally, Method B (Binomial coefficients) is useful only for small σ on this model of machine.

finite computation, using simple arithmetic.

In two dimensions, a similar iterative blurring scheme can be devised, formed from the composition of two one-dimensional blurring steps as given above. Specifically, $h_0(i,j)$ is the initial image data, and $h_N(i,j)$ is obtained from $h_{N-1}(i,j)$ by convolution against the three-by-three mask

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}.$$

Once again, $h_N(i,j)$ will approximate a convolution of the initial data $f(i,j)$ against a Gaussian with $\sigma=\sqrt{N/2}$. In Tables 4, 5, 6, we refer to this method of Gaussian convolution as Method B.

For a serial random access computer (such as a VAX), making use of the separability of symmetry of the kernel (Method A) results in a large savings in terms of number of computations, particularly for large σ. The iterative approach (Method B) is only useful for small σ, where the approximation to a Gaussian will not be as good. Table 4 lists computation counts using a serial random-access machine for different values of σ, with computation Method A, with Method B, and without either method. We have listed arithmetic operation counts separately from accesses to image memory, since different machines may have radically different timings for these two kinds of operations. However, we have lumped together multiplications and additions into the same "arithmetic count" figure.

For a pipeline machine, a critical issue is the size of convolutions that can be performed in a single pass. We assume that addition and scalar multiplication of two images requires a pass through the entire image, but many systems are capable of 3 by 3 convolutions or larger in a single frame time. In Table 5, we list numbers of passes required for Gaussian convolution, considering machines capable of one by one, three by three, and seven by seven convolutions in a single pass. In all cases, we assume that accumulative writes are allowed, i.e., that the result of a local convolution can be summed with an existing image without an additional pass. We've also assumed that scrolling the image data is free, since the shift can usually be combined with another operation by setting offset registers. When these assumptions do not hold, the counts for number of passes will increase. Specifically, if additive writes are not allowed, then operation counts will increase proportionately, since each multiply must be followed by an addition pass. If image scrolling requires a pass (to move the data), then costs are similar to costs for a mesh-connected computer, treated below, where movement of data incurs communication costs.

We see that on a pipeline machine, the iterative method (Method B) is especially useful when three-by-three convolutions are allowed, as long as σ is not too large. If the hardware permits larger single-pass convolutions, a similar iterative method can be defined where each iteration takes advantage of the full available convolution size capabilities.

For a mesh-connected parallel computer, the situation is similar to the pipeline computer case, except that now the number of bits retained in each convolution makes a difference. Further, there may be large differences between bit operations

	$\sigma=1$	$\sigma=2$	$\sigma=4$
Neither A nor B, 4σ cutoff			
1x1	81	289	1089
3x3	18	64	121
7x7	8	9	25
Method A, 4σ cutoff			
1x1	18	34	66
3x3	6	12	22
7x7	4	6	10
Method B			
1x1	12	48	192
3x3	2	8	32
7x7	2	8	32

Table 5. Number of passes required on a pipeline machine for convolution by a Gaussian using various methods of computation. For each method, we list figures for three different architectures: one capable of only image sums and multiplies, one capable of single-pass 3 by 3 convolutions, and one capable of 7 by 7 convolutions.

on the processor and communication costs to access a bit on a neighboring processor. We thus count arithmetic bit operations separate from bit communication counts in Table 6. We assume that the image data is given with 8 bits of precision. For Method A, we assume that the convolution is performed with b bits of precision. For greatest efficiency, the image data is moved among the processors, and the convolution is accumulated in place. For Method B, we consider both the case where b bits are used, and where full precision is retained by allowing the integers to grow as large as needed.

We once again see that Method B, the iterative approach, is less valuable as σ increases. This is because the number of iterations required increases with σ^2. However, since the binomial coefficient approximation to a Gaussian used in Method B is asymptotically accurate as $\sigma \to \infty$, we might well ask whether Method B has *any* utility. One answer is that the utility depends upon the architecture. Certain pipeline machines have multiple stages, and can thus implement multiple iterations rapidly. Accordingly, the range in which the iterative approach is favorable may well include values of σ of interest. A second answer is addressed in Section 5, where we discuss iterative approaches to handling borders.

3. Difference-of-Gaussians

A number of operations are based on difference-of-Gaussian (DoG) or Laplacian-of-Gaussian (LoG) operations. The Marr-Hildreth edge operator is one such; another is the DOLP [5]. First, it should be noted that the DoG is an

	σ=1		σ=2		σ=4	
	ops	com	ops	com	ops	com
Neither A nor B,						
4σ cutoff	729b	648	2601b	2312	9801b	8712
6σ cutoff	1321b	1352	5225b	5000	23409b	20808
Method A						
4σ cutoff	162b	144	306b	272	594b	528
6σ cutoff	234b	208	450b	400	918b	816
Method B						
b bits precision	8b	8b	32b	32b	128b	128b
Full precision	72	56	672	608	8832	8376

Table 6. Number of bit-serial operations (ops) and neighbor bit communication accesses (com) required for Gaussian convolutions on a mesh-connected parallel architecture machine. It is assumed that the image data is stored as 8 bit numbers, one per processor, and that the output convolution values will have b bits of precision (except for Method b, "Full precision" mode). Typically, b will depend on σ, but will generally be at least 16.

approximation to the LoG, in the sense that

$$\Delta G(x,y) = \lim_{\sigma_2 \to \sigma_1} \frac{1/2}{\sigma_1^2 - \sigma_2^2} \Big(G_{\sigma_1}(x,y) - G_{\sigma_2}(x,y) \Big).$$

If σ_1 and σ_2 are very different, then the approximation is not very good. In many schemes, such as the Burt Laplacian pyramid [6], the effective DoG is formed with the ratio of σ_1 and σ_2 fixed. In fact, Marr and Hildreth suggested a separation according to $(\sigma_1/\sigma_2) = 1.6$. This suggestion is often misinterpreted as a statement that the LoG is best approximated by a DoG with $\sigma_1/\sigma_2 = 1.6$, which is clearly not true.

Image 2 shows the zero-crossings at a particular scale for the difference-of-Gaussians applied to the image showed in Image 1, where the ratio of the σ's, $\sigma_1/\sigma_2 = 1.6$, while Image 3 shows the zero-crossings using the Laplacian of the Gaussian using scale σ_2. For Image 2, we used $\sigma_1 = 6.4$ and $\sigma_2 = 4.0$, while the LoG example in Image 3 is based on $\sigma = 4.9$. In this example, we find that the DoG provides slightly better correlation of the zero-crossings with edges and significant features in the original image. We might also compare the DoG with an LoG with a different intermediate value of σ, but our experience with such comparisons is similar to the one shown. Namely, we find that the location and density of zero-crossings is slightly more stable when a DoG is used.

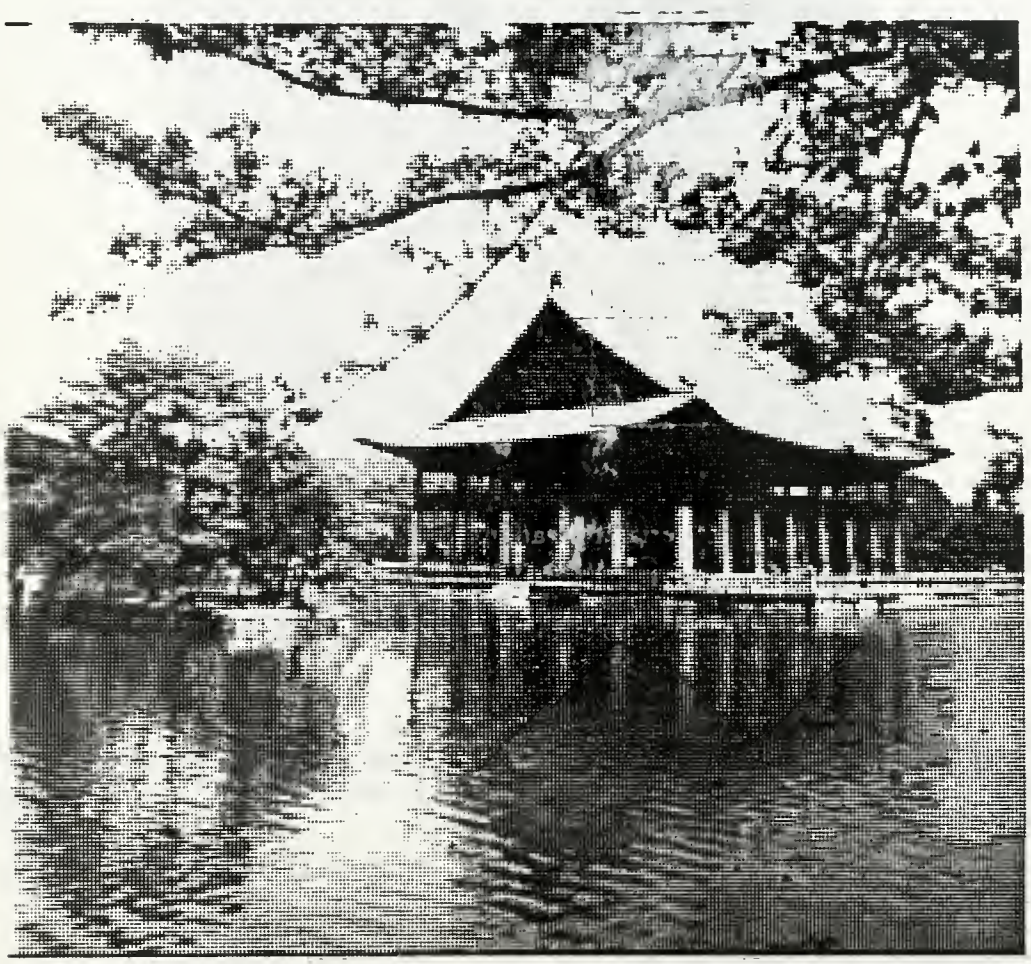


Image 1. An original image, digitized to 512 by 512 pixels.

Image 2. The zero-crossings of the Difference-of-Gaussian filtered version of the image in Image 1, where the Gaussians have standard deviation σ's corresponding to 4 pixels and 6.4 pixels.

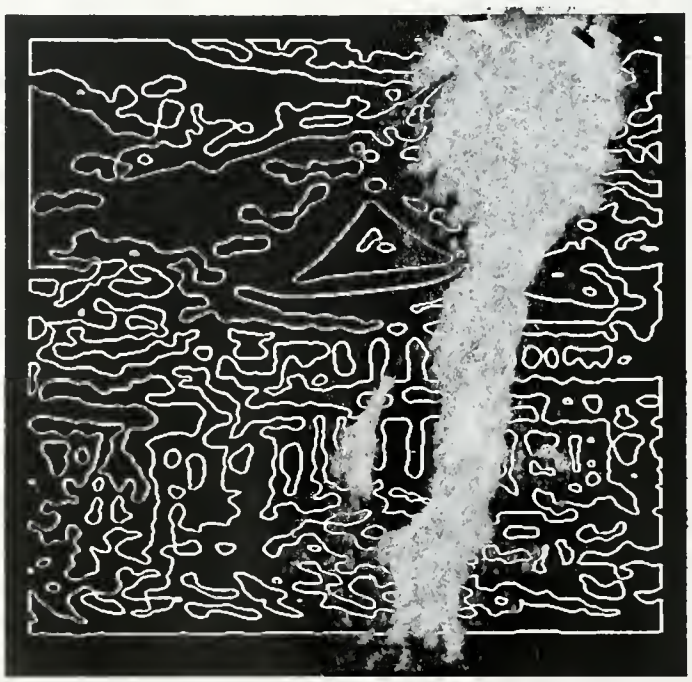


Image 3. The zero-crossings of the Laplacian-of-Gaussian filtered version of the image in Image 1, where the Gaussian has standard deviation σ corresponding to 4.9 pixels.

If a pyramid scheme is desired, then the Burt method is by far the one of choice. The effective ratio of scales, σ_2/σ_1, is equal to 2 in the standard scheme where pyramid levels decrease in size by a factor of 2 between levels. Other schemes, however, are possible [2, 7]. If a DoG is desired at a fixed maximum resolution, then for greatest efficiency, the two Gaussian-convolved images should be computed, and some care should be exercised in computing the difference, since numerical precision may be difficult.

If the LoG is desired, despite advantages of the DoG, it can be calculated by any of the following four methods:

(1) Use the Laplacian-of-Gaussian kernel, averaged in blocks, to convolve against the data $f(i,j)$, as discussed in the previous section. Note that the kernel is not separable, so that the resulting convolution will be expensive.

(2) First compute a Laplacian of the image data, using a four-point Laplacian, i.e.,

$$L(i,j) \doteq f(i-1,j)+f(i+1,j)+f(i,j+1)+f(i,j-1)-4f(i,j),$$

and then Gaussian-blur the result. In this method, numerical precision in

the blurring of $L(i,j)$ is a major concern, since if insufficient bits for the representation are provided, the result of any substantive amount of blurring will be zero.

(3) Compute blurred versions of the data $f(i,j)$ using the functions $h_N(i,j)$ as described before, but using full accuracy by retaining all necessary bits. The LoG is approximated using $h_{N+1} - h_N$.

(4) Use the Huertas-Medioni decomposition of the Laplacian-of-Gaussian into the sum of two separable kernels [8]. Specifically, the decomposition is given by

$$\Delta G(x,y) = C \cdot h_1(x) \cdot h_2(y) + C \cdot h_2(x) \cdot h_1(y),$$

where

$$h_1(x) = \left\{x^2 - \frac{1}{\sigma^2}\right\} \cdot e^{-x^2/2\sigma^2}, \quad h_2(y) = e^{-y^2/2\sigma^2}.$$

Note that in each of the two separable kernels, one of the two dimensions uses a Gaussian convolution. The other convolution in each of the two kernels is a second derivative of a Gaussian in one dimension. The decomposition arises since $\Delta(g(x)g(y)) = g''(x)g(y) + g(x)g''(y)$.

For the examples in Image 2 and 3, we used double-precision floating-point representations of the image and kernel elements, for maximum accuracy. Despite the computational advantages, we did not use Methods A nor B, nor any other special property of the kernels. We used extremely large kernels to essentially eliminate truncation errors. When we used any of the approximation methods or computational speed-ups discussed above on a limited-precision processor, the results of applying the zero-crossings operation were frequently disconcertingly different. The difficulty is that the location of the zero-crossings can be extremely sensitive to small changes in the filtered images. There were always a number of zero-crossings (with high gradients in the filtered data) that were not affected by the approximation methods, but many of the zero-crossings belonging to texture edges and weak edges were considerably changed.

For the computational costs of these methods, one can refer to the appropriate cost of Gaussian blurring. Specifically, Laplacian-of-Gaussian convolution by Method (1) above is equivalent to Gaussian convolution without Method A or B. The appropriate σ cutoff for LoG convolution, however, will be larger than the σ cutoff for Gaussian convolution. Methods (2) and (3) above are essentially Gaussian convolution, using Methods A and B respectively. Finally, the Huertas-Medioni decomposition transforms LoG-convolution into the sum of two separable convolutions, so that the computational times are essentially given by those for Method A, except that the figures should be doubled and an additional sum of two images will be required for the output.

4. Zero-crossings

To compute the zero-crossings of a function of two variables, $h(i,j)$, the data should be thresholded at zero, and border points identified in the resulting binary image. We mention two possible definitions for a border point. A border point can be defined as a pixel that contains a pixel of a different binary value among its four

(or eight) neighbors. In this case, edges will be two pixels thick. Alternatively, a border point can be defined as a "1" pixel with a "0" pixel among its neighbors. One way of finding border pixels using the second definition, appropriate when fast three-by-three convolution hardware is available, is to blur the binary image using the mask with 1's in the eight neighbors and 9 in the center:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 9 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

and then thresholding the result so that values 9 and above, but less than 17, are identified as border pixels. (Here we have used eight pixel neighborhoods.)

Haralick has suggested that zero-crossings of the second directional derivative, where the direction of differentiation is the direction of the local gradient [9], will tend to more accurately locate edges as compared to the Marr-Hildreth edge operator. It should be noted that this operator also yields closed contours for the zero-crossings, and can be computed by finding zero-crossings of:

$$h(x,y) = f_x^2 f_{xx} + 2 f_x f_y f_{xy} + f_y^2 f_{yy},$$

where a subscript denotes a partial derivative. The nonlinear second order partial differential operator on the right hand side is in fact the directional derivative scaled by the square of the gradient, and is thus defined even when the gradient is zero. Canny [10] notes that the operator can be formulated as $(1/2)\nabla f \cdot \nabla |\nabla f|^2$. We used a different computation, formulating the partial derivatives by using local three-by-three convolutions. Specifically we have used:

$$f_x \sim \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad f_y \sim \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & 1 \end{bmatrix},$$

$$f_{xx} \sim \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad f_{xy} \sim \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix}, \quad f_{yy} \sim \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The discretized $h(i,j)$ is a sum of products of local convolutions of image data as given by these (or related) kernels. The image data can be the original image, or an appropriately blurred version of the image data. Haralick recommends that the partial derivatives of f be computed using the "facet model," so that the smoothing is implicit in the representation of the parameters of the local facet for f. Other methods of blurring can be used, including Gaussian blur, although this method of computation does not work well with images that have been substantially blurred by a Gaussian due to the small magnitude of the local gradient of the blurred image. In Image 4a, we show the zero-crossings of the second directional derivative, after the image has been only slightly blurred ($\sigma \approx .7$). This should be compared with zero-crossings of the Laplacian-of-Gaussian of the same image at a comparable scale (Image 4b).

Whether a DoG, LoG, or second directional derivative is used, the zero-crossings do not necessarily track the edges accurately. Prominent edges will generally have a zero-crossing curve associated with them (at some scale of resolution),

but the zero-crossings will tend to wander off of the edges and complete loops for which the main edge for s only a portion of the loop. Thus it is desired to tag pixels identified as a zero-c ssing with a measure of the edge strength. One way of doing this, used frequent in practice, is to measure the slope of the filtered image at the zero-crossing. Th edges usually give rise to zero-crossings for which the crossing has a c Thus a measure of importance is given by $|\nabla(\Delta G*f)|(x,y)$, at p ere $\Delta G*f(x,y) = 0$. This idea, for example, is mentioned in Marr's bo slope at a zero-crossing can be computed by taking a Sobel gradient ma of the Laplacian-of-Gaussian filtered image. We have found that by throwing out ero-crossing points where the Sobel gradient magnitude is below a threshold, the remaining curves form a much more accurate depiction of the edges in the image. Image 5 shows the zero-crossings filtered by the Sobel gradient, corresponding to the unfiltered zero-crossings shown in Image 3.

Image 4a. The zero-crossings of the second directional derivative, computed in the direction of the gradient (the Haralick operator). In regions where the gradient is zero, the operator returns zero, and thus zero-crossings do not occur, since the border points are defined as pixels that have a positive value with a nonpositive value in the neighborhood. The original image was slightly blurred by a Gaussian. The effective σ is roughly 0.7 pixels.

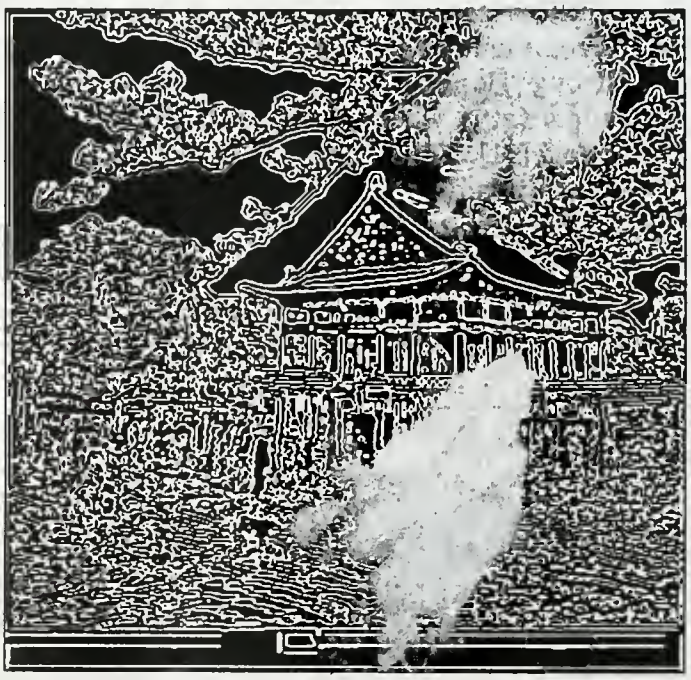


Image 4b. The zero-crossings of a Laplacian-of-Gaussian convolution of the image, taken at approximately the same scale of resolution as Image 4a. We note that the Laplacian-of-Gaussian tends to introduce a number of extraneous zero-crossings, and tends to be slightly less accurate near corners.

Image 5. The zero-crossings of Image 3, with a brightness encoding of the Sobel gradient magnitude of the Laplacian of the Gaussian of the image data at those zero-crossings.

Here, we have simply displayed the zero-crossings with the intensity proportional to the magnitude of the Sobel gradient at the zero-crossing. When this image is thresholded, the result is curves (which are no longer closed contours) with some of the weak edges removed.

Alternatively, it is possible to "and" the results of finding zero-crossing contours at several successive scales of resolution. Providing the contours are thickened in all but one of the scales before the "and"-ing, the result again is a collection of curves, not necessarily closed, but associated with prominent edges. Marr and Hildreth speculated on methods for combining zero-crossings from multiple channels (scales), and conjunctive combination was one suggestion [1]. In essence,

"and"-ing zero-crossings at successive levels demands that zero-crossings not move rapidly as the scale changes. Rapid movement is possible only if the gradient at the zero-crossing is small, although it can happen that [illegible]-crossings with low gradient also survive "and"-ing at several levels.

5. Borders

All of the convolutions discussed so far [illegible] the original data $f(x,y)$ is defined for $(x,y)\in\mathbb{R}^2$ and that the grid of [illegible] is an infinite lattice. Generally, however, images are defined on a finite [illegible]angular grid of points, or sometimes on an irregularly shaped grid. Suppose that $f(i,j)$ is defined for $(i,j)\in\mathcal{D}$, where $\mathcal{D}$ is a domain of grid points. We define the boundary points of the domain as points in $\mathcal{D}$ with at least one of the eight neighbors outside of $\mathcal{D}$, and denote those points by $\partial\mathcal{D}$. How should the infinite convolutions be defined?

A usual tactic is to simply extend $f(i,j)$ to [illegible]ero for pixels outside of $\mathcal{D}$. For Gaussian convolution, however, there are [illegible]nate approaches, motivated by regarding Gaussian convolution as a mean [illegible] solving the Heat equation [12]. Specifically, we can use the formulas [illegible]iterative blurring in conjunction with fixed boundary data:

$$\begin{aligned}16h_{N+1}(i,j) &= h_N(i-1,j-1)+2h_N(i-1,j)+h_N(i-1,j+1)\\ &+2h_N(i,j-1)+4h_N(i,j)+2h_N(i,j+1)\\ &+h_N(i+1,j-1)+2h_N(i+1,j)+h_N(i+1,j+1)\end{aligned}$$

for $(i,j)\in\mathcal{D}-\partial\mathcal{D}$, and $h_{N+1}(i,j) = h_N(i,j)$ for $(i,j)\in\partial\mathcal{D}$. Although this yields something different from Gaussian blurring, the result is an appropriate blurring of the image data. An analog to difference-of-Gaussian filters can then be computed. Note that such difference filters, when defined this way, will always give zero values on the boundary.

Another possibility for handling borders is to, in essence, insist that the normal derivative at the borders be zero. In terms of the discrete blurring kernels, this condition can be implemented (for a convex grid of points $\mathcal{D}$, and when the kernel is no bigger than three-by-three) by the following rule:

Every access to a pixel outside of the grid $\mathcal{D}$ should substitute for that pixel the value of the closest pixel in the grid $\mathcal{D}$.

Thus, for example, suppose that $\mathcal{D}$ consists of the rectangular array of pixels $\{(i,j) \mid 0 \le i \le 511,\ 0 \le j \le 511\}$. Then to update a pixel on the left edge, $(i,0)$, with $1 \le i \le 510$ (not a corner point), the formula is

$$\begin{aligned}h_{N+1}(i,0) = (1/16)\cdot\Big[&3h_N(i-1,0) + h_N(i-1,1)\\ &+ 6h_N(i,0) + 2h_N(i,1) + 3h_N(i+1,0) + h_N(i+1,1)\Big].\end{aligned}$$

Similarly, in the upper left corner, the update rule is

$$h_{N+1}(0,0) = (1/16)\cdot\Big[9h_N(0,0) + 3h_N(0,1) + 3h_N(1,0) + h_N(1,1)\Big].$$

For such a rectangular grid $\mathcal{D}$, the rule can be seen to be equivalent to first blurring the rows by the formulas using the masks

$\begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix}$ at interior points,

$\begin{bmatrix} 0 & 3/4 & 1/4 \end{bmatrix}$ at left edges,

$\begin{bmatrix} 1/4 & 3/4 & 0 \end{bmatrix}$ at right edges,

and then blurring the resulting columns by similar masks oriented vertically.

This method of updating has the property that the global mean is constant, i.e., $\sum_i \sum_j h_N(i,j)$ is independent of N. Once again, differences of levels can be formed to construct analogs to the DoG and LoG structure. Image 6a shows the finite rectangular image of Image 1 blurred by this method. This blurring can be compared to the image that results if the original image is blurred on an infinite domain, and then restricted to the original size (Image 6b).

Acknowledgements

This research was supported by Office of Naval Research Grant N00014-85-K-0077, Work Unit NR 4007006, and NSF grants DCR-8403300 and DCR-8502009.

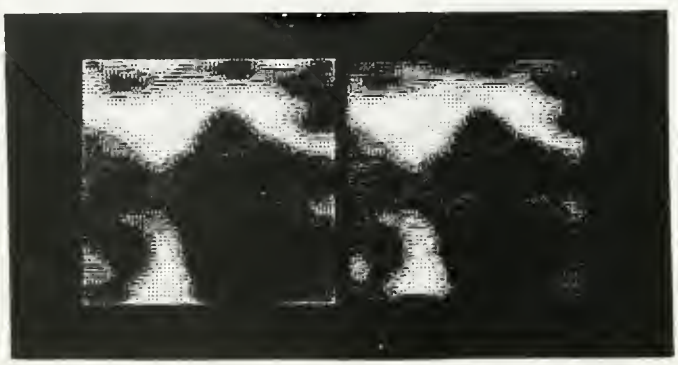


Image 6. The left image (Image 6a) is a 128 by 128 image blurred by the iterative method, keeping the normal derivative at the borders equal to zero. The average intensity value is kept constant by this method. $N = 32$ stages of blurring were used, corresponding to $\sigma = 4$ in a Gaussian convolution. On the right (Image 6b), the same image is blurred an equivalent amount by the iterative method, assuming that pixels outside of the image domain have value zero. The average intensity value drops as the number of blurring iterations increases when using this method. All difference will occur within 32 pixels of the border.

References

[1] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings Royal Society London (B)*, p. 187 (1980).

[2] P. Burt, "Fast filter transforms for image processing," *Computer Graphics and Image Processing* 16, p. 20 (1981).

[3] A. Huertas and G. Medioni, "Detection of intensity changes with subpixel accuracy using Laplacian-Gaussian masks," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8, pp. 651-664 (1986).

[4] E. C. Hildreth, "Implementation of a theory of edge detection," MIT Technical Report AI-TR-579 (April, 1980). Master's Thesis.

[5] James L. Crowley and Alice C. Parker, "A representation for shape based on peaks and ridges in the difference of low-pass transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6, pp. 156-169 (1984).

[6] P. Burt and T. Adelson, "The laplacian pyramid as a compact image code," *IEEE Trans. on Communications*, p. 532 (1983).

[7] Shmuel Peleg, Orna Federbusch, and Robert Hummel, *Custom-made pyramids*, Submitted.

[8] J. S. Chen, A. Huertas, and G. Medioni, "Very fast convolution with Laplacian-of-Gaussian masks," *Proceedings of the Conference on Computer Vision and Pattern Recogntion*, pp. 293-298 IEEE, (June, 1986).

[9] R. Haralick, "Digital step edges for zero crossings of second directional derivatives," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, p. 58 (1984).

[10] J. F. Canny, "Finding edges and lines in images," MIT AI Lab Technical Report 720 (1983).

[11] D. Marr, *Vision*, W. H. Freeman and Company (1982).

[12] Robert Hummel, "Representations based on zero-crossings in scale-space," *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pp. 204-209 (June, 1986).

NYU COMPSCI TR-254 c.1
Hummel, Robert
Computing large-kernel
convolutions of images.